National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir

Submitted by:

Eman Babar

Reg. number:

23-NTU-CS-FL-1148

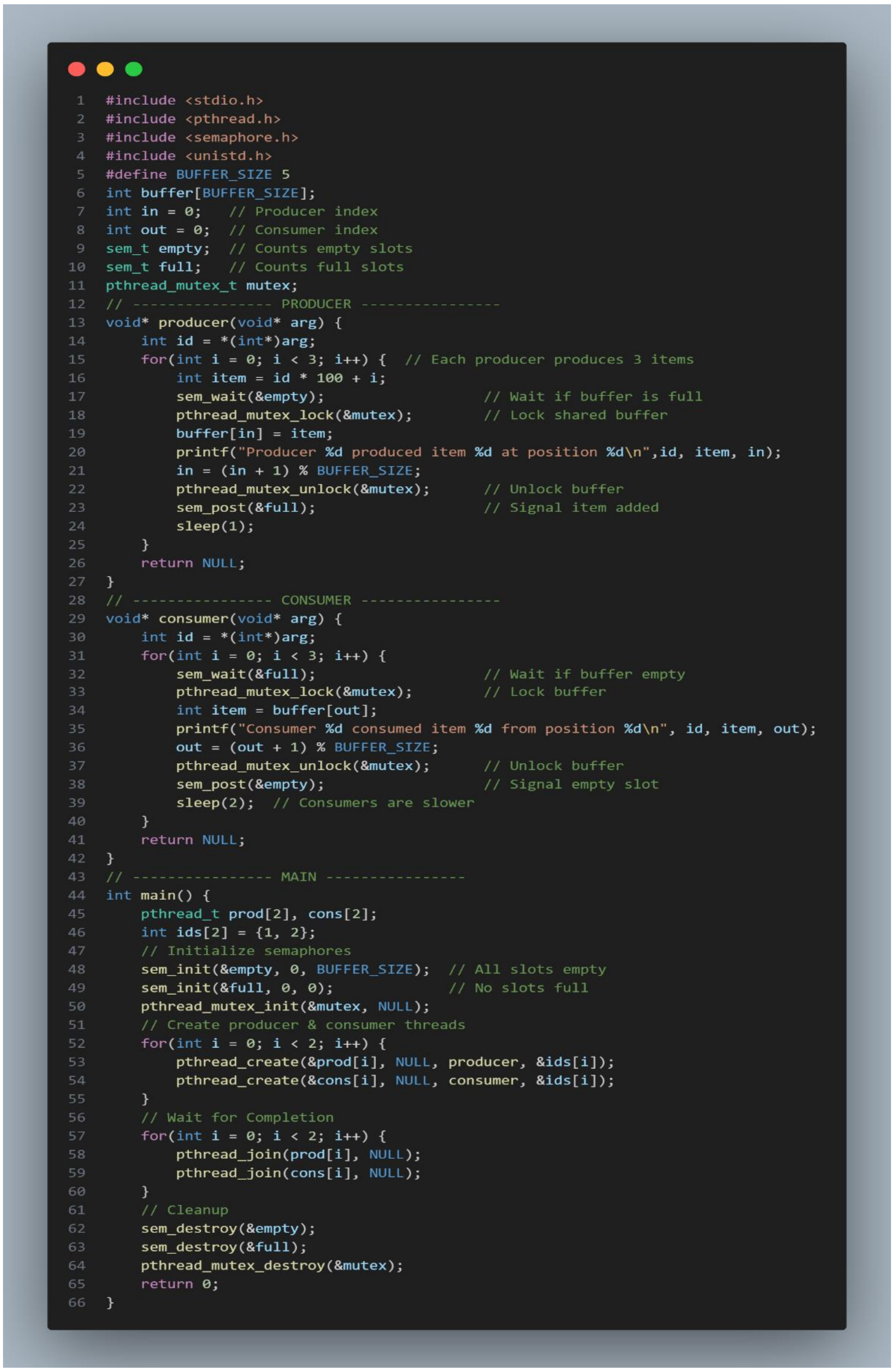
Semester:

5th- A

LAB-10

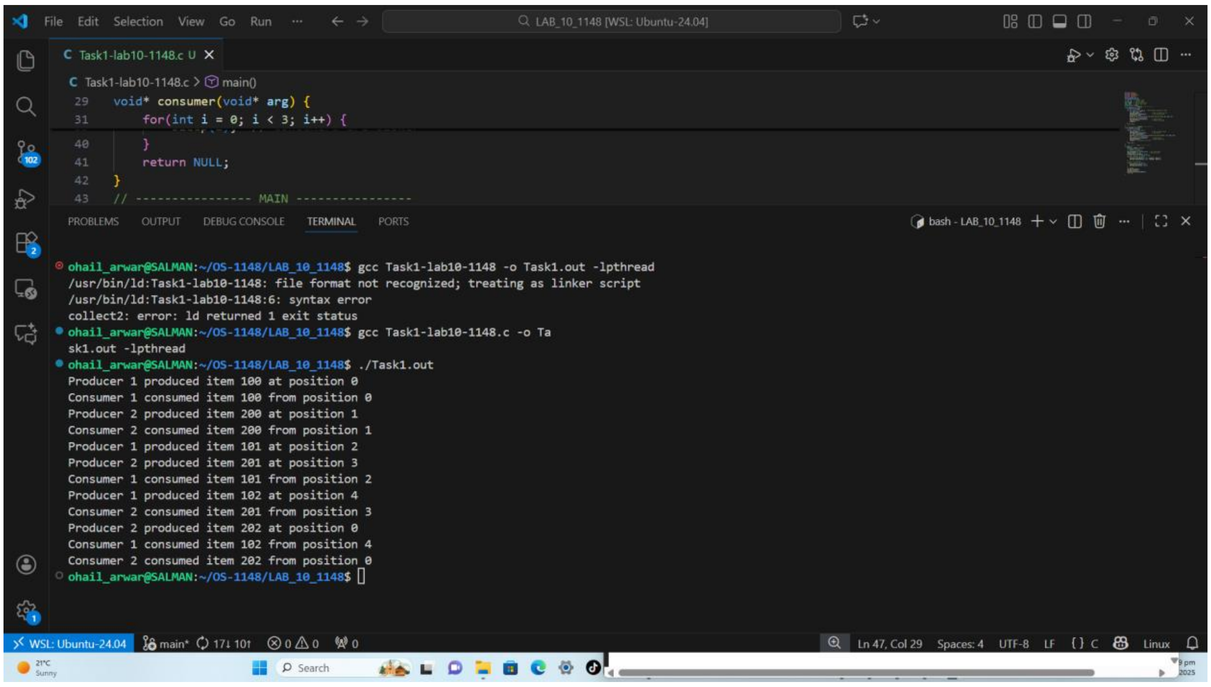
Task1: Producer-Consumer Problem

Code:



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0;    // Producer index
int out = 0;  // Consumer index
sem_t empty;  // Counts empty slots
sem_t full;   // Counts full slots
pthread_mutex_t mutex;
// ---------------- PRODUCER ----------------
void* producer(void* arg) {
    int id = *(int*)arg;
    for(int i = 0; i < 3; i++) {  // Each producer produces 3 items
        int item = id * 100 + i;
        sem_wait(&empty);                   // Wait if buffer is full
        pthread_mutex_lock(&mutex);         // Lock shared buffer
        buffer[in] = item;
        printf("Producer %d produced item %d at position %d\n",id, item, in);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);       // Unlock buffer
        sem_post(&full);                    // Signal item added
        sleep(1);
    }
    return NULL;
}
// ---------------- CONSUMER ----------------
void* consumer(void* arg) {
    int id = *(int*)arg;
    for(int i = 0; i < 3; i++) {
        sem_wait(&full);                    // Wait if buffer empty
        pthread_mutex_lock(&mutex);         // Lock buffer
        int item = buffer[out];
        printf("Consumer %d consumed item %d from position %d\n", id, item, out);
        out = (out + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);       // Unlock buffer
        sem_post(&empty);                   // Signal empty slot
        sleep(2);  // Consumers are slower
    }
    return NULL;
}
// ---------------- MAIN ----------------
int main() {
    pthread_t prod[2], cons[2];
    int ids[2] = {1, 2};
    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE);  // All slots empty
    sem_init(&full, 0, 0);             // No slots full
    pthread_mutex_init(&mutex, NULL);
    // Create producer & consumer threads
    for(int i = 0; i < 2; i++) {
        pthread_create(&prod[i], NULL, producer, &ids[i]);
        pthread_create(&cons[i], NULL, consumer, &ids[i]);
    }
    // Wait for Completion
    for(int i = 0; i < 2; i++) {
        pthread_join(prod[i], NULL);
        pthread_join(cons[i], NULL);
    }
    // Cleanup
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Output:



```c
void* consumer(void* arg) {
    for(int i = 0; i < 3; i++) {
    }
    return NULL;
}
// --------------- MAIN ---------------
```

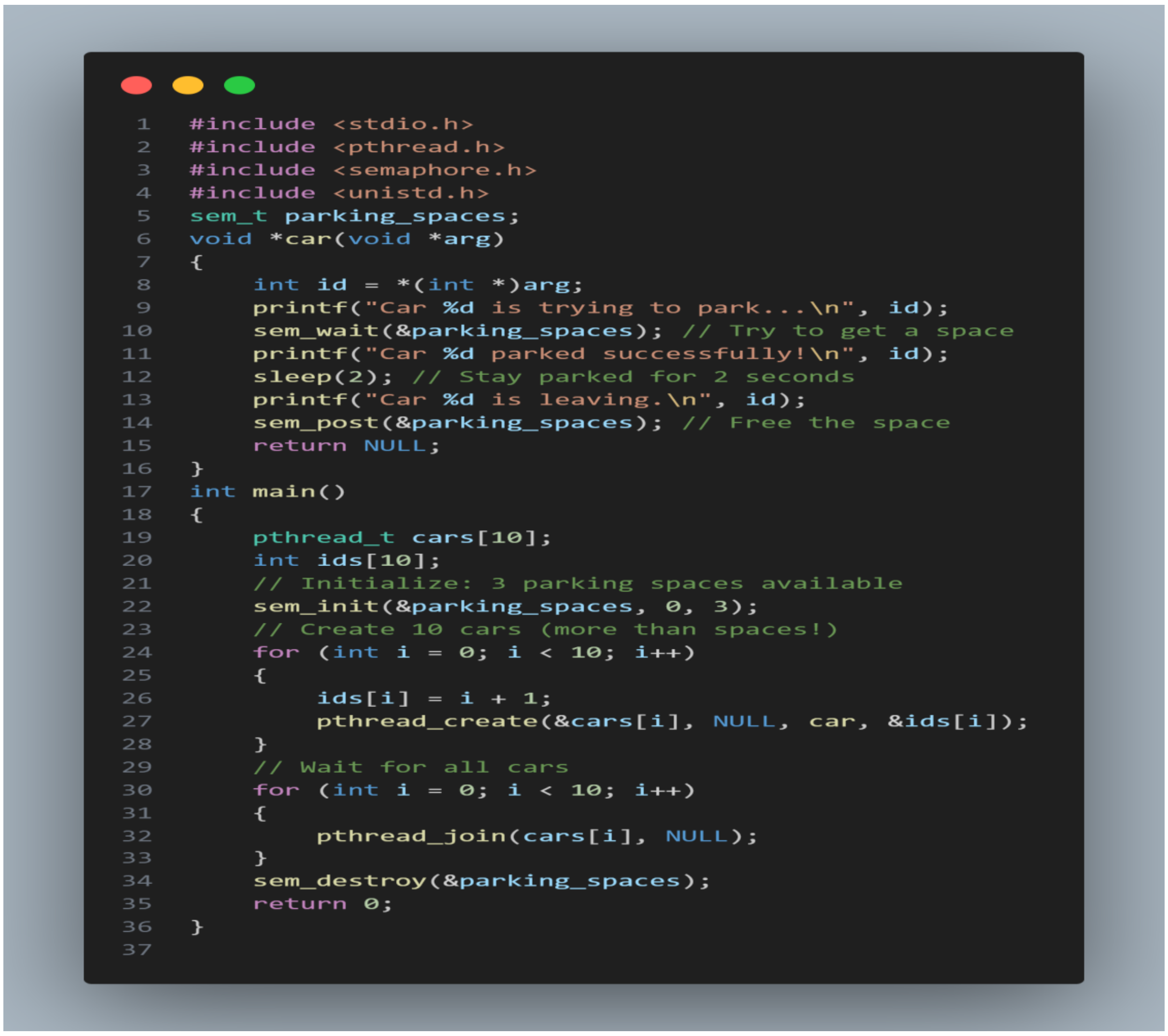
```
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ gcc Task1-lab10-1148 -o Task1.out -lpthread
/usr/bin/ld:Task1-lab10-1148: file format not recognized; treating as linker script
/usr/bin/ld:Task1-lab10-1148:6: syntax error
collect2: error: ld returned 1 exit status
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ gcc Task1-lab10-1148.c -o Ta
sk1.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ ./Task1.out
Producer 1 produced item 100 at position 0
Consumer 1 consumed item 100 from position 0
Producer 2 produced item 200 at position 1
Consumer 2 consumed item 200 from position 1
Producer 1 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Consumer 1 consumed item 101 from position 2
Producer 1 produced item 102 at position 4
Consumer 2 consumed item 201 from position 3
Producer 2 produced item 202 at position 0
Consumer 1 consumed item 102 from position 4
Consumer 2 consumed item 202 from position 0
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ 
```

Remarks:

in it, two producer threads generate items and place them into a shared buffer, while two consumer threads remove and also process those items. The “**empty”** and **“full”** semaphores control buffer capacity that prevents overfilling and also underflow. A mutex lock ensures that only one thread accesses the buffer at a time.

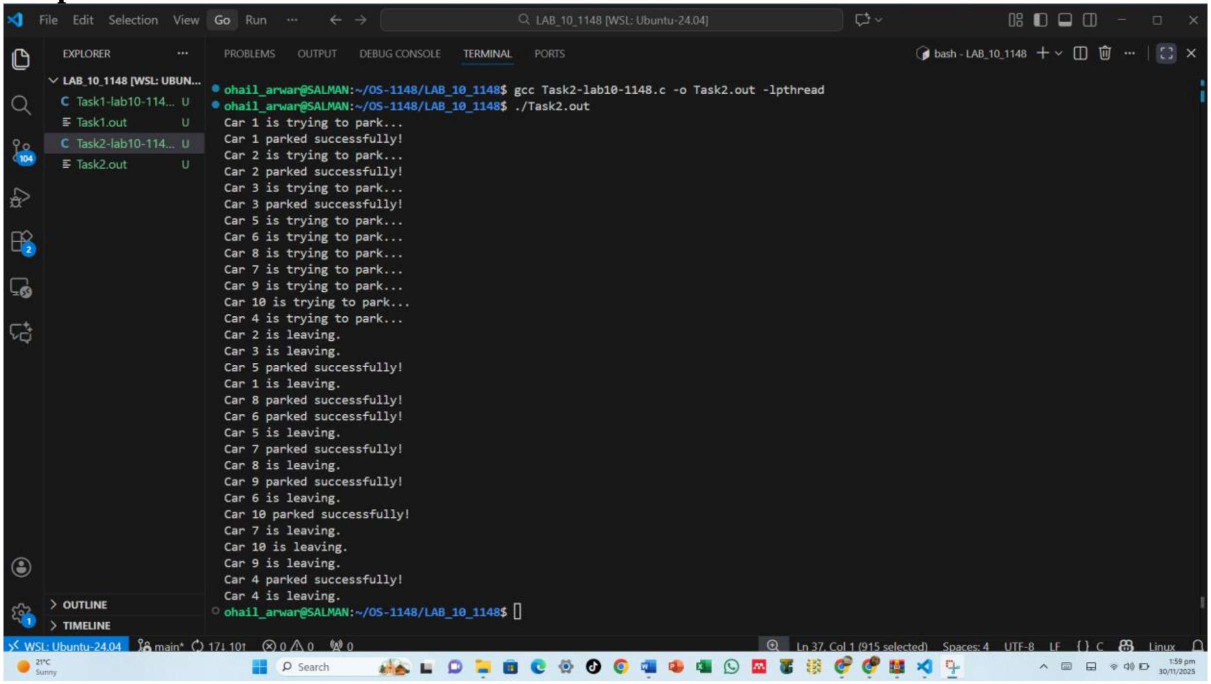
Task2: Producer-Consumer Problem by Counting Semaphore

Code:



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t parking_spaces;
void *car(void *arg)
{
    int id = *(int *)arg;
    printf("Car %d is trying to park...\n", id);
    sem_wait(&parking_spaces); // Try to get a space
    printf("Car %d parked successfully!\n", id);
    sleep(2); // Stay parked for 2 seconds
    printf("Car %d is leaving.\n", id);
    sem_post(&parking_spaces); // Free the space
    return NULL;
}
int main()
{
    pthread_t cars[10];
    int ids[10];
    // Initialize: 3 parking spaces available
    sem_init(&parking_spaces, 0, 3);
    // Create 10 cars (more than spaces!)
    for (int i = 0; i < 10; i++)
    {
        ids[i] = i + 1;
        pthread_create(&cars[i], NULL, car, &ids[i]);
    }
    // Wait for all cars
    for (int i = 0; i < 10; i++)
    {
        pthread_join(cars[i], NULL);
    }
    sem_destroy(&parking_spaces);
    return 0;
}
```

Output:



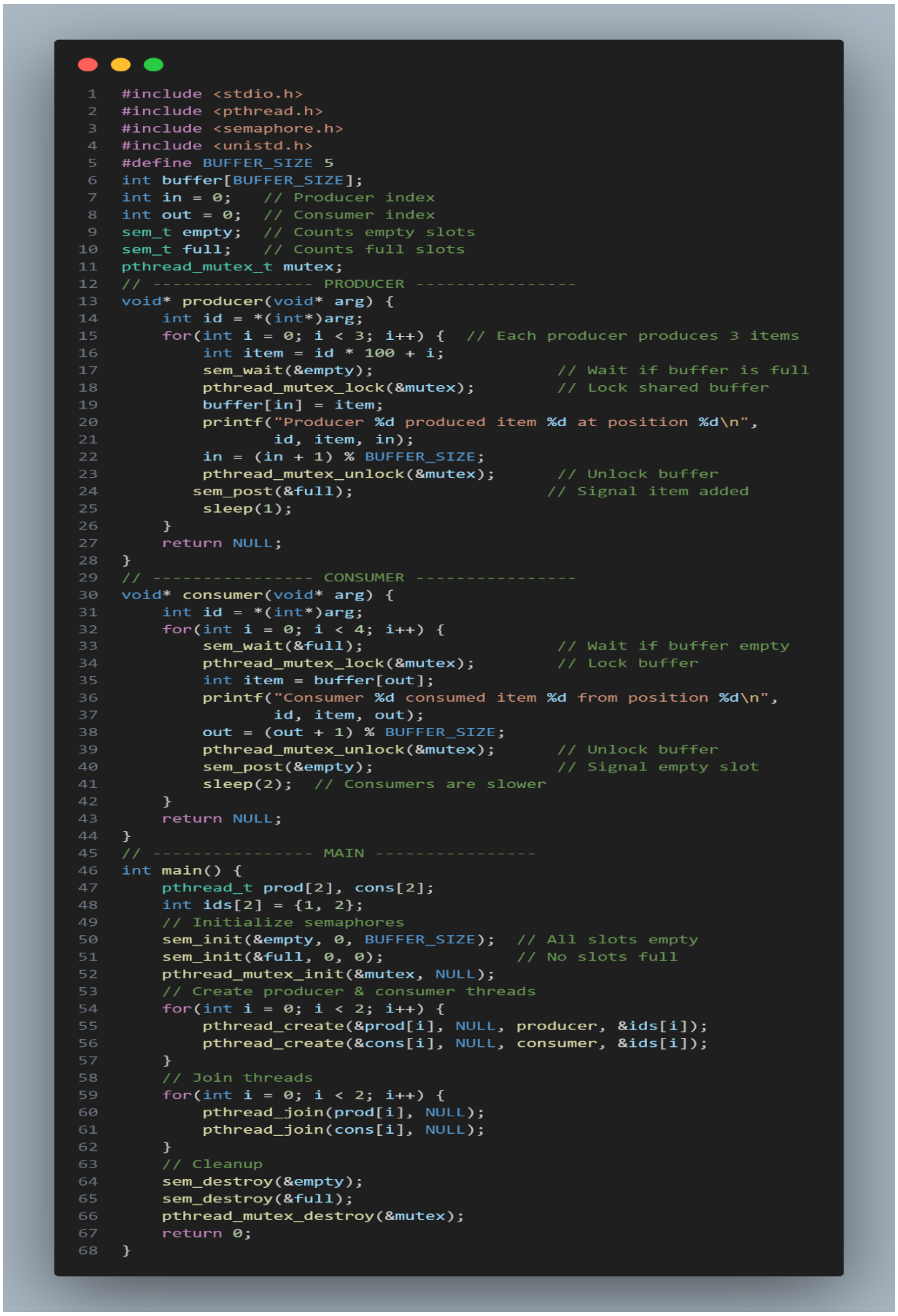
```
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ gcc Task2-lab10-1148.c -o Task2.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ ./Task2.out
Car 1 is trying to park...
Car 1 parked successfully!
Car 2 is trying to park...
Car 2 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 5 is trying to park...
Car 6 is trying to park...
Car 8 is trying to park...
Car 7 is trying to park...
Car 9 is trying to park...
Car 10 is trying to park...
Car 4 is trying to park...
Car 2 is leaving.
Car 3 is leaving.
Car 5 parked successfully!
Car 1 is leaving.
Car 8 parked successfully!
Car 6 parked successfully!
Car 5 is leaving.
Car 7 parked successfully!
Car 8 is leaving.
Car 9 parked successfully!
Car 6 is leaving.
Car 10 parked successfully!
Car 7 is leaving.
Car 10 is leaving.
Car 9 is leaving.
Car 4 parked successfully!
Car 4 is leaving.
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$
```

Remarks:

In this program, a semaphore **"parking_spaces"** is initialized with 3 spaces. It means that only three cars can park at a time. Ten threads of cars are created, and each car tries to park by calling **“sem_wait”,** which blocks if all the spaces are full. Cars can get a space to park for 2 seconds, then leave and call **‘sem_post”** to free the space. At last, the main thread waist for all car threads to finish and destroys the semaphore.

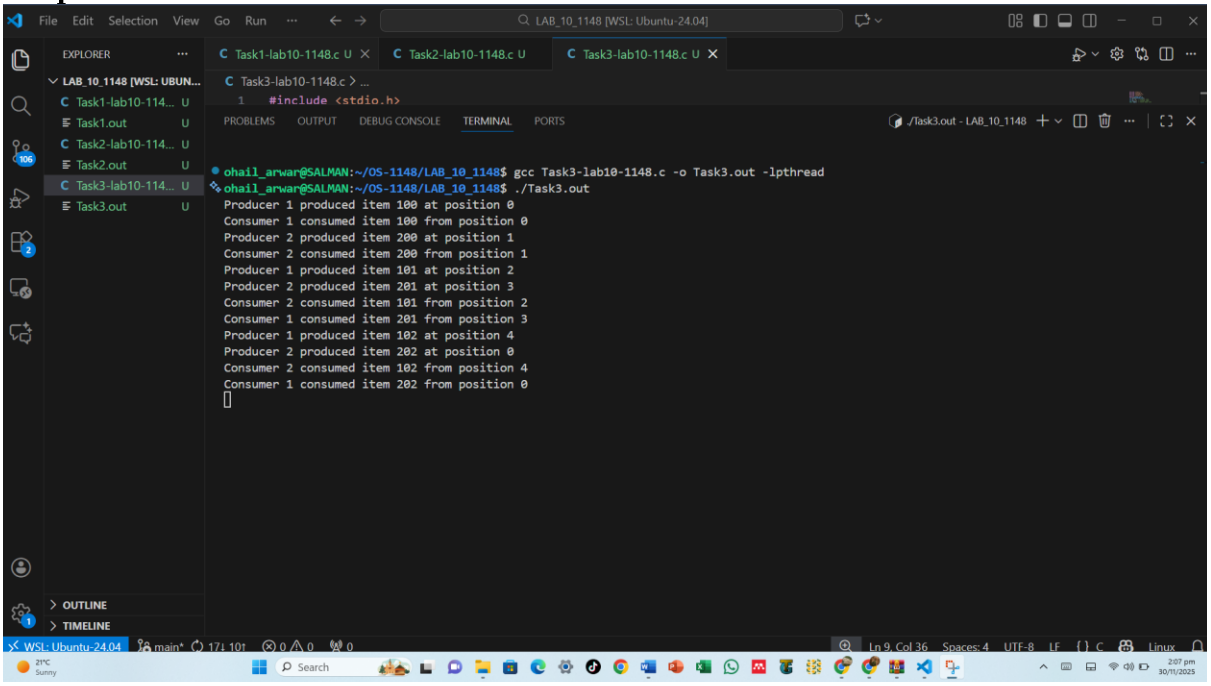
<u>Task3: Producer-Consumer Problem (Block Condition)</u>

Code:



```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0;   // Producer index
int out = 0;  // Consumer index
sem_t empty;  // Counts empty slots
sem_t full;   // Counts full slots
pthread_mutex_t mutex;
// ---------------- PRODUCER ----------------
void* producer(void* arg) {
    int id = *(int*)arg;
    for(int i = 0; i < 3; i++) {  // Each producer produces 3 items
        int item = id * 100 + i;
        sem_wait(&empty);                  // Wait if buffer is full
        pthread_mutex_lock(&mutex);        // Lock shared buffer
        buffer[in] = item;
        printf("Producer %d produced item %d at position %d\n",
               id, item, in);
        in = (in + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);      // Unlock buffer
       sem_post(&full);                    // Signal item added
        sleep(1);
    }
    return NULL;
}
// ---------------- CONSUMER ----------------
void* consumer(void* arg) {
    int id = *(int*)arg;
    for(int i = 0; i < 4; i++) {
        sem_wait(&full);                   // Wait if buffer empty
        pthread_mutex_lock(&mutex);        // Lock buffer
        int item = buffer[out];
        printf("Consumer %d consumed item %d from position %d\n",
               id, item, out);
        out = (out + 1) % BUFFER_SIZE;
        pthread_mutex_unlock(&mutex);      // Unlock buffer
        sem_post(&empty);                  // Signal empty slot
        sleep(2);  // Consumers are slower
    }
    return NULL;
}
// ---------------- MAIN ----------------
int main() {
    pthread_t prod[2], cons[2];
    int ids[2] = {1, 2};
    // Initialize semaphores
    sem_init(&empty, 0, BUFFER_SIZE);  // All slots empty
    sem_init(&full, 0, 0);             // No slots full
    pthread_mutex_init(&mutex, NULL);
    // Create producer & consumer threads
    for(int i = 0; i < 2; i++) {
        pthread_create(&prod[i], NULL, producer, &ids[i]);
        pthread_create(&cons[i], NULL, consumer, &ids[i]);
    }
    // Join threads
    for(int i = 0; i < 2; i++) {
        pthread_join(prod[i], NULL);
        pthread_join(cons[i], NULL);
    }
    // Cleanup
    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Output:



```
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ gcc Task3-lab10-1148.c -o Task3.out -lpthread
ohail_arwar@SALMAN:~/OS-1148/LAB_10_1148$ ./Task3.out
Producer 1 produced item 100 at position 0
Consumer 1 consumed item 100 from position 0
Producer 2 produced item 200 at position 1
Consumer 2 consumed item 200 from position 1
Producer 1 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Consumer 2 consumed item 101 from position 2
Consumer 1 consumed item 201 from position 3
Producer 1 produced item 102 at position 4
Producer 2 produced item 202 at position 0
Consumer 2 consumed item 102 from position 4
Consumer 1 consumed item 202 from position 0
```

Remarks:

In this program, producer inserts items into a shared buffer while consumer removes them. Semaphores **“empty”** and **“full”** control buffer availability. Each producer generates 3 items and each consumer consumes 4 items. This program ensures synchronization and avoid race conditions and buffer overflow.